Cika Electrónica

Nota de Aplicación: CAN-032
Título: **Utilización de displays LCD color con controladores S1D13705 y Rabbit**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	15/06/05	

En la CAN-031 les presentamos los nuevos displays color LCD de 320x240. Dijimos que se trata de displays sin controlador, por lo que su utilización dentro de un sistema requiere de un controlador externo, o de la generación de todas las señales de control por parte del usuario. Veremos en esta nota la forma de utilizar un controlador de Epson, el S1D13705, para que se encargue de todo lo relacionado con el manejo del display, mientras que nosotros nos limitaremos a decirle cómo lo tiene que hacer, y darle la información a mostrar.

Breve descripción del S1D13705

Hardware
El S1D13705 es un controlador inteligente para displays LCD de alta resolución, permitiendo no sólo displays color pasivos sino también blanco y negro y TFT. El S1D13705 se encarga de generar las señales que necesita el display, como viéramos en la CAN-031, tanto para este tipo de display como para los otros mencionados, según como esté configurado. La configuración de este dispositivo es algo compleja, pero afortunadamente el fabricante nos provee un software que nos permite, a partir de los valores operacionales del display, generar los datos a escribir en los diferentes registros. La imagen a enviar al display se aloja en una RAM interna de 80KB (81920 bytes), la cual es direccionada de forma lineal por diecisiete líneas de address, y accesible mediante un bus de datos de 16-bits. En la parte superior del mapa de memoria, se encuentran los treinta y dos registros de control (0x1FFE0) que nos permiten controlar el funcionamiento a voluntad. La interfaz entre el S1D13705 y el procesador host puede elegirse entre una variedad de procesador específicos soportados, y dos modos genéricos, uno de los cuales emplearemos en la presente aplicación, mediante pines destinados a tal fin. El arbitraje de la memoria entre la generación de la imagen en el display y el acceso por parte del procesador se resuelve mediante la petición de *wait-states* al procesador. Puede funcionar tanto con procesadores y displays de 3,3 como de 5V, mientras que el core funciona a 3,3V

Software
El S1D13705 se encarga de todo lo referente al despliegue de la imagen, la cual reside como dijéramos en su memoria interna (80KB). Mediante los registros de control, es posible indicar en qué zona de memoria comienza la pantalla y su tamaño, así como también la cantidad de bits asignados a definir el color de cada pixel, lo cual definirá la distribución de la memoria. Para 8bpp, cada byte representa un pixel y los mismos se distribuyen de arriba a abajo y de izquierda a derecha, conforme avanzan las posiciones de memoria. Para 4bpp, es similar, pero se empaquetan dos pixels por byte y el pixel de la izquierda ocupa el nibble más significativo. Es posible almacenar varias pantallas en memoria y cambiar la posición de inicio, siempre y cuando, claro está, su tamaño y resolución permitan que quepa.
La cantidad de colores determinada es un índice dentro de una paleta RGB 4:4:4, es decir, 4096 colores posibles, y el controlador maneja todo esto de forma transparente, el programador especifica el modo de operación, carga la paleta, y trabaja con los índices, similar a como se haría con cualquier pantalla o formato de archivo en modo indexado (indexed-color).

Desarrollo propuesto

Para mantener la simpleza, optamos por mantenernos dentro de un mapa de 64K para el direccionamiento de memoria. Al direccionar registros, lo indicamos mediante una llamada especial para operar sobre A16. Esto limita la capacidad de almacenamiento interno de pantallas, pero permite simplificar el software de bajo nivel, albergando una pantalla completa de 320x240 pixels en 4bpp (16 colores por pixel).

Algoritmos

Para ubicar un punto en pantalla, calculamos su posición en memoria sabiendo que alojamos dos pixels por byte, es decir: $mem = \frac{x}{2} + 160 \cdot y$. El resto de $\frac{x}{2}$ nos dirá qué nibble utilizar

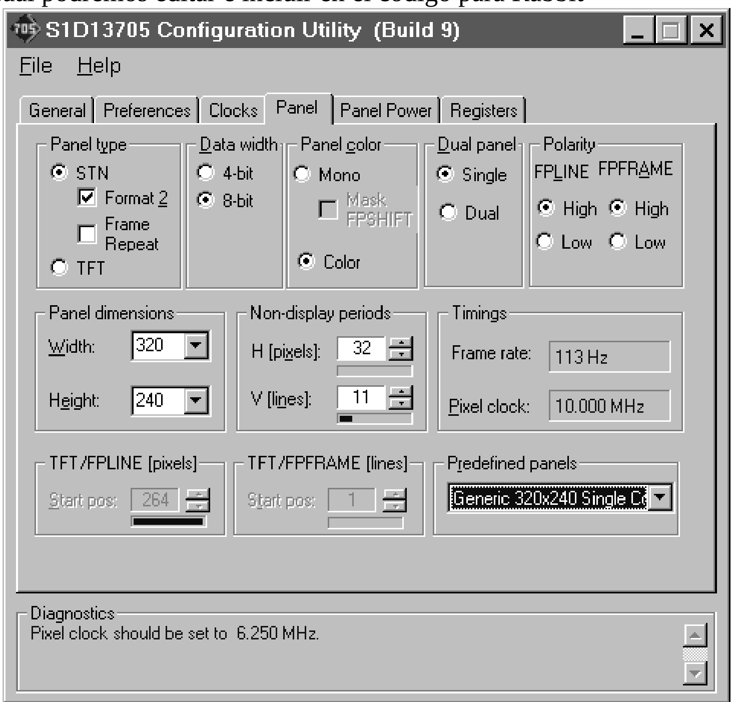
Para graficar funciones, debemos tener en cuenta que la coordenada (0;0) se halla en el extremo superior izquierdo de la pantalla.

Para mostrar pantallas, deberemos agrupar los datos de modo tal de poder enviarlos de forma que aproveche de manera eficiente la estructura de memoria. Si comparamos la estructura de memoria del display con la forma de guardar imágenes en 16 colores en formato BMP, veríamos que son muy similares, por ejemplo: BMP va de abajo a arriba y el display de arriba a abajo, por lo que la imagen se ve espejada verticalmente. Además, BMP incluye un encabezado que contiene la paleta de colores.

Por consiguiente, para adaptar una imagen, debemos llevarla a la resolución deseada, reducirla a 16 colores, espejarla verticalmente, salvarla en formato BMP y por último descartar los 118 bytes del comienzo con algún editor hexa. Entre los bytes a descartar tomaremos los bytes 54 a 117, los cuales corresponden a la paleta en formato BGR0 (4 bytes), y la guardaremos como RGB.

Configuración del S1D13705

Para obtener los valores a setear en cada uno de los registros, utilizamos el software de configuración provisto por el fabricante, definiendo un **display genérico color de un sólo panel STN, de 320x240 pixels, con datos en formato 2**. La frecuencia de reloj corresponderá al utilizado, en este caso 20 MHz, el cual haremos que se divida a la mitad, para mantener las frecuencias de operación dentro de los rangos recomendados por el fabricante del display (solapa *clocks*). Recordemos indicar 4bpp en la solapa *Preferences* El resto de los parámetros podemos dejar los valores por defecto, o leer detenidamente el manual técnico del S1D13705 para saber cómo configurar a nuestro antojo. Exportamos los datos en un archivo de tipo *C header file* (s1d13705.h), el cual podremos editar e incluir en el código para Rabbit



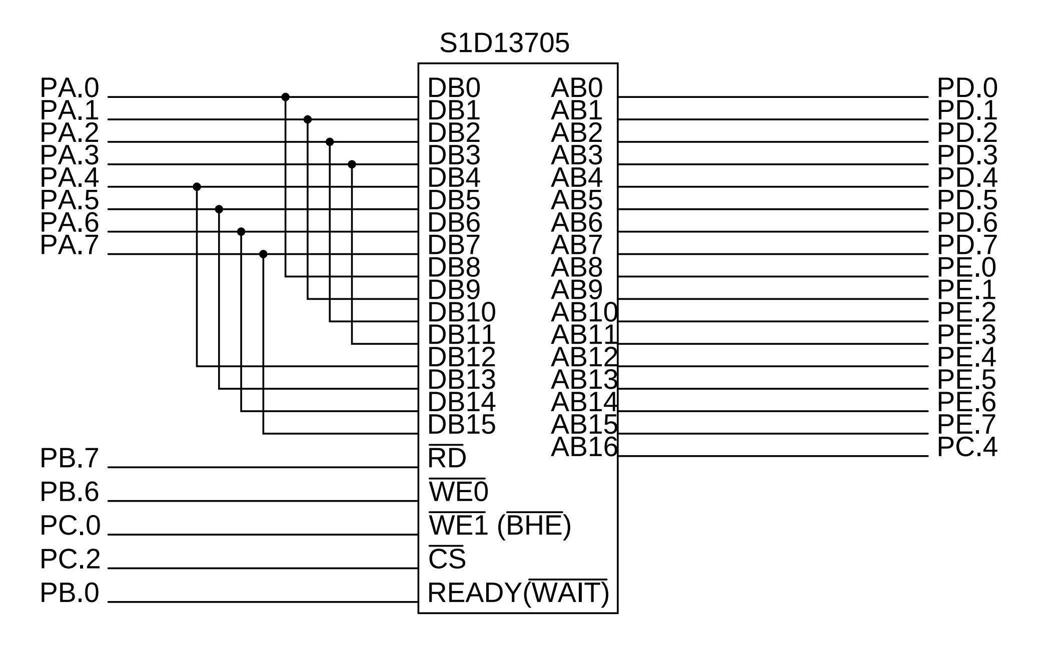
Hardware de interfaz

Si bien el S1D13705 puede funcionar a 3 ó 5V, hemos elegido para esta nota un RCM2100, y operaremos a 5V. No podemos emplear el bus de Rabbit para agilizar la operación, dado que debemos respetar la petición de espera cuando el controlador nos lo indique, y Rabbit no tiene línea de $\overline{WAIT}$, *READY*, o equivalente, sino

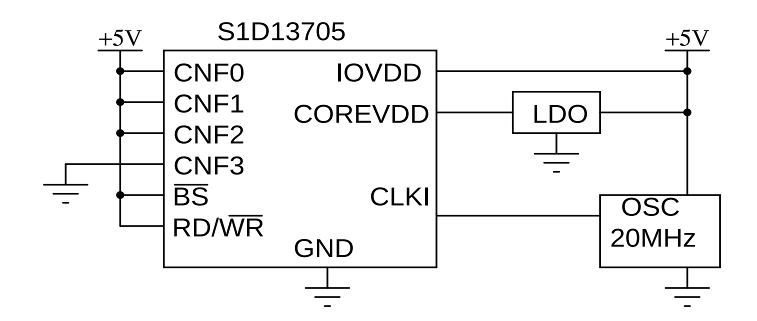
que genera una inserción automática de wait-states por configuración. En este caso, el controlador no especifica una duración máxima del ciclo y debemos respetar la petición de espera. Desarrollaremos entonces el hardware empleando los ports de I/O, uno de cuyos pines será conectado a la línea de $\overline{WAIT}$ del controlador; generaremos las direcciones pertinentes mediante I/O también. En cuanto al bus de 16-bits, lo que haremos es duplicar nuestros 8-bits conectando parte alta y parte baja del bus entre sí, elegimos para esta nota el modo de operación "Generic #2", y le indicamos al controlador si la operación es en la parte baja o la parte alta del bus (a la usanza del 8086). De este modo, la información siempre sale del Rabbit por los mismos ocho pines, y el controlador la lee por los ocho altos o los ocho bajos, según indique la señal $\overline{BHE}$ (Bus High-byte Enable), que no es otra cosa que la inversión de A0.

Entre las limitaciones de cantidad de pines y posibilidad de operación, elegimos algunos de ellos con funciones similares dentro del mismo port, a fin de agilizar las rutinas de manejo. Queda como tarea para el lector, el estudio de la posibilidad de realizarlo con otros pines e incluso emplear algún otro de los modos de interfaz del S1D13705; para los alcances de esta Nota de Aplicación, la performance obtenida es satisfactoria.

El diagrama siguiente indica el esquema de conexión propuesto:

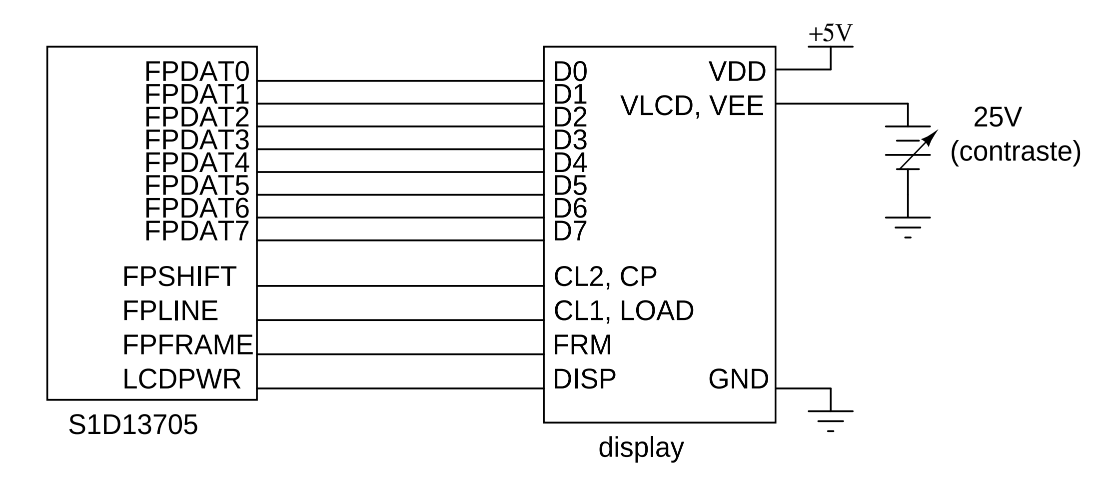


La selección del tipo de interfaz se realiza mediante los pines *CNFx*. El diagrama de configuración, alimentación, y misceláneos, es el siguiente:



Hardware de display

La conexión al display se realiza mediante las líneas analizadas en la CAN-031, como indica el diagrama a continuación:



La tensión de contraste es conveniente que sea variable para poder ajustarlo, podemos utilizar un step-up como el MC34063A (comercializado por Cika) y controlarla mediante un preset a modo de divisor resistivo, o proveerla de forma externa.

Software de bajo nivel

Tenemos un bus de 16-bits simulado dentro de uno de 8-bits, con señalización similar a la utilizada por el 8086, es decir, cuando operamos sobre el byte "alto" (parte "alta" del bus, D8 a D15), lo indicamos activando la señal $\overline{BHE}$. Luego de activar $\overline{CS}$ e indicar la operación, observamos la señal $\overline{WAIT}$ para ver si podemos seguir adelante con la operación o debemos esperar. A los fines prácticos incluimos toda la operación de generación de líneas de direcciones en una subrutina. Dado que tendremos bastantes datos para escribir, necesitamos hacerlo rápido, razón por la cual utilizamos assembler para las rutinas críticas.

```
#asm root
read13705::
       call addressbus              ; pone addresses y BHE
       ld hl,PCDR                   ; apunta a port paralelo
       ioi res 2,(HL)               ; baja CS
       ld hl,PBDR                   ; apunta a port paralelo
       ioi res 7,(HL)               ; baja RD
rwait:
       ioi ld a,(hl)                ; lee port paralelo
       rrca                         ; chequea bit 0 (WAIT)
       jr nc,rwait                  ; espera hasta que NOWAIT
       ioi ld a,(PADR)              ; lee dato
       ioi set 7,(HL)               ; sube RD
       ld hl,PCDR                   ; apunta a port paralelo
       ioi set 2,(HL)               ; sube CS
       ld h,0
       ld l,a
       ret

write13705::
       call addressbus              ; put addresses and BHE
       ld hl,PCDR                   ; apunta a port paralelo
       ioi res 2,(HL)               ; baja CS
       ld hl,(sp+4)                 ; lee dato
       ld a,l
       ioi ld (PADR),a              ; pone dato
       ld a,0x84
       ioi ld (SPCR),a              ; A= salidas
       ld hl,PBDR                   ; apunta a port paralelo
       ioi res 6,(HL)               ; baja WE0
wwait:
```

```
        ioi ld a,(hl)               ; lee port paralelo
        rrca                        ; chequea bit 0 (WAIT)
        jr nc,wwait                 ; espera hasta que NOWAIT
        ioi set 6,(HL)              ; sube WE0 ( >1/BCLK luego de NOWAIT)
        ld hl,PCDR                  ; apunta a port paralelo
        ioi set 2,(HL)              ; sube CS
        ld a,0x80
        ioi ld (SPCR),a             ; A= entradas
        ret

addressbus:
        ld hl,(sp+4)                ; lee address
        ex de,hl                    ; en DE
ab2:    ld hl,PEDR                  ; apunta a port
        ioi ld (hl),d               ; pone parte alta
        ld hl,PDDR                  ; apunta a port
        ioi ld (hl),e               ; pone parte baja
        ld hl,PCDR                  ; apunta a port paralelo
        ioi set 0,(hl)              ; BHE=1
        rrc e                       ; debo activar BHE ? (test LSB=1)
        jr nc,ok1                   ; no
        ioi res 0,(HL)              ; sí, BHE=0
ok1:
        ret
```

Para poder observar algo, deberemos definir los colores, cargando la paleta. Eso lo hacemos mediante la siguiente rutina:

```
void writepalette(triplet *address)
{
int i;
unsigned char *ptr;

        ptr=&(address->RGB[0]);
        BitWrPortI(PCDR,&PCDRShadow,1,4);                           // A16=1
        write13705(0x15+S1DREGOFFSET,0);
        for(i=0;i<16;i++){
                write13705(S1DREGOFFSET+0x17,*ptr++);
                write13705(S1DREGOFFSET+0x17,*ptr++);
                write13705(S1DREGOFFSET+0x17,*ptr++);
        }
        BitWrPortI(PCDR,&PCDRShadow,0,4);                           // A16=0
}
```

Cada paleta estará definida como un array de tres elementos, o triplet, definiendo los valores correspondientes a R, G, y B. Si bien el controlador utiliza sólo 4-bits, definimos un byte para cada color por simplicidad y semejanza con VGA.

```
typedef union {
        unsigned char RGB[3];
} triplet;

const triplet VGApalette[16]= {
{       0x00, 0x00, 0x00},          /* BLACK  */
{       0x00, 0x00, 0xA0},          /* BLUE   */
{       0x00, 0xA0, 0x00},          /* GREEN  */
{       0x00, 0xA0, 0xA0},          /* CYAN   */
{       0xA0, 0x00, 0x00},          /* RED    */
{       0xA0, 0x00, 0xA0},          /* PURPLE */
{       0xA0, 0xA0, 0x00},          /* YELLOW */
{       0xA0, 0xA0, 0xA0},          /* WHITE  */
{       0x00, 0x00, 0x00},          /* BLACK  */
{       0x00, 0x00, 0xF0},          /* LT BLUE   */
{       0x00, 0xF0, 0x00},          /* LT GREEN  */
{       0x00, 0xF0, 0xF0},          /* LT CYAN   */
{       0xF0, 0x00, 0x00},          /* LT RED    */
{       0xF0, 0x00, 0xF0},          /* LT PURPLE */
{       0xF0, 0xF0, 0x00},          /* LT YELLOW */
{       0xF0, 0xF0, 0xF0}           /* LT WHITE  */
};
```

A continuación, la inicialización del chip. Los valores los obtuvimos utilizando el software de configuración provisto por el fabricante, según comentáramos.

```
typedef unsigned char S1D_INDEX;
typedef unsigned char S1D_VALUE;
typedef struct
{
    S1D_INDEX Index;
    S1D_VALUE Value;
} S1D_REGS;

const static S1D_REGS aS1DRegs[] =
{
    { 0x00,             0x24 },       // Revision Code Register
    { 0x01,             0x3B },       // Mode Register 0 Register
    { 0x02,             0xB0 },       // Mode Register 1 Register
    { 0x03,             0x03 },       // Mode Register 2 Register (enable display)
    { 0x04,             0x27 },       // Horizontal Panel Size Register
    { 0x05,             0xEF },       // Vertical Panel Size Register (LSB)
    { 0x06,             0x00 },       // Vertical Panel Size Register (MSB)
    { 0x07,             0x1F },       // FPLINE Start Position Register
    { 0x08,             0x00 },       // Horizontal Non-Display Period Register
    { 0x09,             0x00 },       // FPFRAME Start Position Register
    { 0x0A,             0x0B },       // Vertical Non-Display Period Register
    { 0x0B,             0x00 },       // MOD Rate Register
    { 0x0C,             0x00 },       // Screen 1 Start Address Register (LSB)
    { 0x0D,             0x00 },       // Screen 1 Start Address Register (MSB)
    { 0x0E,             0x00 },       // Screen 2 Start Address Register (LSB)
    { 0x0F,             0x00 },       // Screen 2 Start Address Register (MSB)
    { 0x10,             0x00 },       // Screen Start Address Overflow Register
    { 0x11,             0x00 },       // Memory Address Offset Register
    { 0x12,             0xFF },       // Screen 1 Vertical Size Register (LSB)
    { 0x13,             0x03 },       // Screen 1 Vertical Size Register (MSB)
    { 0x18,             0x00 },       // GPIO Configuration Control Register
    { 0x19,             0x02 },       // GPIO Status/Control Register
    { 0x1A,             0x00 },       // Scratch Pad Register
    { 0x1B,             0x00 },       // SwivelView Mode Register
    { 0x1C,             0xA0 },       // Line Byte Count Register
};

#define S1DNUMREGS 25
#define S1DREGOFFSET 0xFFE0
#define S1DMEMSIZE 38400

void init13705()
{
int i;

       BitWrPortI(PCDR,&PCDRShadow,1,4);                            // A16=1
       for(i=1;i<S1DNUMREGS;i++)
              write13705((aS1DRegs[i].Index)+S1DREGOFFSET,aS1DRegs[i].Value);
       BitWrPortI(PCDR,&PCDRShadow,0,4);                            // A16=0
       writepalette(VGApalette);
}
```

Software

El resto del software lo escribimos mayormente en C, por comodidad y velocidad de desarrollo. Se trata de simples y comunes rutinas que no incluiremos aquí para no extender el texto, pero que el lector puede obtener del archivo adjunto con el software, o consultar en cualquiera de las otras notas de aplicación, dado que son muy similares.

Una excepción es el volcado de imágenes, para el cual escribimos una función especial en assembler. Dado el tamaño de una imagen ($\frac{320\times240}{2}=38400\,bytes$), necesitamos una función rápida que copie la imagen directamente desde xmem, leyendo por words y escribiendo de a dos bytes en el S1D13705. Básicamente, modificamos el código de *xmem2root()* y lo adaptamos a nuestros propósitos. En la CAN-029 encontrarán una rutina más simple, pero que hace uso de una función externa que resuelve el mapeo de dirección absoluta en

20-bits a valores de *XPC* y dirección lógica. Este ejemplo es una muestra de cómo suceden las cosas a nivel assembler, directamente.

```
;IY: address
;(IX+0): data left
;(IX+1): data right
writew13705::
       ld hl,iy                     ; get A15-A0
       ex de,hl                     ; in DE
       call ab2                     ; put addresses and BHE
       ld hl,PCDR                   ; apunta a port paralelo
       ioi res 2,(HL)               ; baja CS
       ld a,(IX+0)                  ; lee dato
       ioi ld (PADR),a              ; pone dato
       ld a,0x84
       ioi ld (SPCR),a              ; A= salidas
       ld hl,PBDR                   ; apunta a port paralelo
       ioi res 6,(HL)               ; baja WE0
wwait1:
       ioi ld a,(hl)                ; lee port paralelo
       rrca                         ; chequea bit 0 (WAIT)
       jr nc,wwait1                 ; espera hasta que NOWAIT
       ioi set 6,(HL)               ; sube WE0 ( >1/BCLK después de NOWAIT)
       ld hl,PCDR                   ; apunta a port paralelo
       ioi set 2,(HL)               ; sube CS
       ld hl,iy                     ; lee address
       inc hl                       ; inc address
       ex de,hl                     ; A15-A0 in DE
       call ab2                     ; address bus
       ld hl,PCDR                   ; apunta a port paralelo
       ioi res 2,(HL)               ; baja CS
       ld a,(IX+1)                  ; lee dato
       ioi ld (PADR),a              ; pone dato
       ld a,0x84
       ioi ld (SPCR),a              ; A= salidas
       ld hl,PBDR                   ; apunta a port paralelo
       ioi res 6,(HL)               ; baja WE0
wwait2:
       ioi ld a,(hl)                ; lee port paralelo
       rrca                         ; chequea bit 0 (WAIT)
       jr nc,wwait2                 ; espera hasta que NOWAIT
       ioi set 6,(HL)               ; sube WE0 ( >1/BCLK después de NOWAIT)
       ld hl,PCDR                   ; apunta a port paralelo
       ioi set 2,(HL)               ; sube CS
       ld a,0x80
       ioi ld (SPCR),a              ; A= entradas
       ret


xdumpblk::
               ld      a,xpc                ; save the frame pointer, and xpc
               push    af
               push    ix

               ld      de,0xe000            ; de=lsw(dest)-0xe000
               ld      hl,(sp+8)
               or      a
               sbc     hl,de
               ex      de,hl
               ld      hl,(sp+9)            ; hl:e=(dest-0xe000)>>8
               jr      nc,.xcbf_1
               dec     h
.xcbf_1:
               ld      l,d

               add     hl,hl                ; h=(dest-0xe000)>>12
               add     hl,hl
               add     hl,hl
               add     hl,hl
               ld      a,h
               ld      xpc,a                ; xpc=(dest-0xe000)>>12

               ex      de,hl                ; calculate the offset in xpc window
               ld      a,h                  ;    ix=(dest&0x0fff)|0xe000
```

```
               and     0x0f
               or      0xe0
               ld      h,a
               ld      ix,hl

               ld      iy,(sp+6)              ; a:iy=dest, hl=length
               ld      hl,(sp+12)

                                              ; count words not bytes
               ld      c,h                    ; c:b=length
               ld      b,l
               ld      a,l
               or      a
               jr      nz,xcbf_loop
               dec     c

xcbf_loop:
               call    writew13705            ; preserve BC
               ld      de,0x0002              ; 6 copying word by word
               add     ix,de                  ; 4
               add     iy,de                  ; 4
               djnz    xcbf_loop              ; 5, critical path word
               xor     a
               dec     c
               jp      p,xcbf_loop

xcbf_done:
               pop     ix
               pop     af
               ld      xpc,a
               ret
#endasm

root useix void LCD_dump(long imgdata, triplet *paldata)
{

unsigned int dest,tocopy,len;

        len=S1DMEMSIZE/2;
        dest=0;
        imgdata+=sizeof(long);
        writepalette(paldata);
        while(len) {
                tocopy=2048-(dest&2047);
                if(tocopy>len)
                        tocopy=len;
                xdumpblk(dest,imgdata,tocopy);
                dest+=(tocopy<<1);
                imgdata+=(tocopy<<1);
                len-=tocopy;
        }
}
```

Como punto importante, tengamos en cuenta al inicializar el módulo de setear correctamente los pines bidireccionales en el sentido en que los usamos, y todos en el estado inactivo. Inmediatamente después, inicializamos el chip:

```
        WrPortI ( SPCR, &SPCRShadow, 0x80 );                // PA0-7 = Inputs
        WrPortI ( PEDDR,&PEDDRShadow,'\B11111111' );
        WrPortI ( PDDDR,&PDDDRShadow,'\B11111111' );
        WrPortI ( PBDR,&PBDRShadow,'\B11000000' );
        WrPortI ( PCDR,&PCDRShadow,'\B11111111' );
        init13705();
```

Nota importante

Existe un inconveniente en el arranque, que es el hecho de que el Rabbit puede poner en estado activo varias de las señales de control, generando contenciones en el bus del S1D13705 y causando excesivo consumo de corriente. Esta situación se prolonga hasta tanto el Rabbit es correctamente inicializado y dependiendo del tipo de fuente de alimentación utilizada puede dificultar el arranque o tal vez causar daño a alguno de los chips. Se

recomienda, en un entorno de producción, limitar estas condiciones. Una posibilidad es impedir que la señal $\overline{CS}$ se active hasta tanto los pines de I/O han sido correctamente inicializados, por ejemplo, mediante un buffer adicional con una constante RC en el pin de habilitación.